What is claimed is:

1. A method for programming a multithreaded application so as to minimize thread switching overheads and memory usage during processing of the application, the method comprising:

   a. writing a plurality of errands, the errands being functions performing specific tasks that collectively constitute the entire thread functionality; and

   b. forming one or more itineraries corresponding to a thread, the itineraries controlling the execution of the errands in the desired manner.

2. The method as recited in claim 1 wherein an itinerary is shared by multiple threads within the multithreaded application.

3. The method as recited in claim 1 wherein forming the itinerary comprises:

   a. storing information regarding the errands that need to be executed, the information being stored in the form of pointers to the actual errand functions, the pointers being stored in an errand function list;

   b. storing the sequence in which the errands are to be executed, the sequence being the order in which the errand function pointers are stored within the errand function list; and

   c. storing data required for execution of the errands within the itinerary, the data being stored in an errand data list.

4. The method as recited in claim 3 wherein forming the itinerary further comprises defining an errand state field, the errand state field capable of being modified by preemptive errands, the preemptive errands being functions that may cause the thread to temporarily stall further execution, the value assigned to the errand state field at any point of thread execution being indicative of the state of the thread.

5. A method for executing a multithreaded application, the multithreaded application having been programmed using itinerarized threads, the itinerarized threads

comprising standard thread constructs as well as itineraries, the itineraries being lists of errands, the errands being functions performing specific tasks, the method comprising:

a. compiling the application code;

b. scheduling the plurality of threads for execution;

c. executing threads running standard thread constructs in normal mode, the normal mode of execution being thread execution in accordance with standard thread execution methodology, the normal-mode execution of a thread being carried out using an execution stack associated with the thread;

d. executing itineraries in itinerary mode, the itinerary mode being a special thread execution scheme for executing the complete itinerary, the itinerary mode execution of a thread being carried out using a kernel stack;

e. exiting the itinerary mode of execution when the complete itinerary corresponding to a thread has been executed, the thread being subsequently executed in normal mode.

6. The method as recited in claim 5 wherein itinerary execution is carried out after preempting the thread running in normal mode in response to a request for running the itinerary on behalf of the thread.

7. The method as recited in claim 5 wherein scheduling the plurality of threads comprises:

a. sequentially queuing the plurality of threads for resource allocation based on predefined criteria, the queuing being done for allocation of each resource to the threads; and

b. allocating a resource to the threads in accordance with the queue maintained for the resource.

8. The method as recited in claim 7 wherein executing threads in the normal mode comprises:

   a. loading the thread context, the thread context being information required for execution of the thread;

   b. executing thread specific logic; and

   c. preempting the thread in response to a request for the thread preemption, the thread preemption being done after storing the thread context in the execution stack associated with the thread and storing a pointer to the execution stack in the thread structure.

9. The method as recited in claim 5 wherein executing threads in the itinerary mode comprises:

   a. setting up the itinerary, the setting up of the itinerary comprising:

      i. setting up the kernel stack, the kernel stack being set up in such a manner that it becomes the execution stack for the itinerary;

      ii. setting up an errand state variable, the errand state variable being subsequently used for ascertaining the state of the thread running the errand;

      iii. setting up an errand function list, the errand function list storing pointers to the actual errand functions;

      iv. setting up an errand data list, the errand data list storing data required for execution of the errands; and

      v. storing pointers to the errand function list and errand data list, the pointers being used subsequently for accessing the lists; and

   b. executing the errands corresponding to the itinerary sequentially, the sequence of the errands being specified through the errand function list, the errands being executed using data stored in the errand data list.

10. The method as recited in claim 9 wherein the sequence of executing the errands is modified by conditional errands, conditional errands being errands that change the sequence of errand execution based upon certain predefined conditions.

11. The method as recited in claim 9 wherein executing errands sequentially comprises:

while the complete itinerary is not executed, iteratively performing the following steps:

a. loading an errand for execution, the errand being loaded in accordance with the specified sequence of errands in the errand function list;

b. running the errand and receiving the return value returned by the errand, a true return value indicating that the errand was executed successfully, a false return value indicating that the errand blocked and did not complete successfully;

c. executing the next errand in the errand function list in response to a true return value returned by the errand being executed;

d. blocking the itinerary in response to a false return value returned by the errand being executed, the blocking being done in the itinerary mode; and

e. resuming itinerary execution when the blocked itinerary is scheduled back, the itinerary execution being resumed from the errand that had caused the itinerary to block.

12. The method as recited in claim 11 wherein running an errand comprises:

a. checking the value of the errand state field of the itinerary, the errand state field being checked for ascertaining whether the errand is being executed for the first time or it is a previously blocked errand, the errand returning a true value if it is a previously blocked errand;

b. executing the complete errand in case the errand is being executed for the first time;

c. returning a true return value if the complete errand is executed successfully; and

d. modifying the errand state field and returning a false return value if the errand is blocked, the errand state field being modified to indicate that the errand is being blocked.

13.A multithreaded application processing system executing a multithreaded application program, the program comprising a plurality of standard threads and itinerarized threads, the standard threads having been written using standard thread methodology, the itinerarized threads comprising standard thread constructs and itineraries, the itineraries being lists of errands, the errands being functions performing specific tasks, the system comprising:

a. a compiler compiling the application program;

b. a memory storing information related to the threads, the memory comprising:

i. a plurality of thread stacks, each of the thread stacks being associated with a thread, the stack being stored with context information pertaining to the thread, the context information being the information set required for processing of the thread; and

ii. a kernel stack, the kernel stack being used by the itinerarized threads while the itineraries are being processed;

c. a processor processing the plurality of threads, the processor accessing the memory for information pertaining to the plurality of threads; and

d. an operating system scheduling and managing execution of the plurality of threads, the operating system executing the standard threads and standard thread constructs of itinerarized threads in accordance with standard thread execution methodology, the operating system executing itineraries in itinerary mode, the itinerary mode being a thread execution scheme for executing the complete itinerary.

14. The system as recited in claim 13 wherein the multithreaded processing system utilizes multiple processors, each of the multiple processors having a separate kernel stack associated with it in the memory.

15. The system as recited in claim 13 wherein the operating system comprises:

a. a scheduler scheduling various standard and itinerarized threads for execution;

b. a standard thread running service executing various standard threads and standard thread constructs of itinerarized threads; and

c. an itinerary running service executing the itineraries corresponding to the itinerarized threads in itinerary mode.

16. The system as recited in claim 15 wherein the scheduler comprises:

a. a ready queue for holding the plurality of threads ready for execution, the plurality of threads being standard as well as itinerarized threads ready for execution;

b. means for prioritizing the plurality of threads on the basis of pre-defined criteria; and

c. means for scheduling the plurality of threads in order of priority as defined by the pre-defined criteria.

17. The system as recited in claim 16 wherein the scheduler maintains separate ready queues for holding the standard and itinerarized threads.

18. The system as recited in claim 15 wherein the standard thread running service comprises:

a. a preemption service enabling thread preemption;

b. standard preemptive services, the preemptive services being specific activities that might preempt the thread, the standard preemptive services using the preemption service for execution; and

c. standard non-preemptive services, the non-preemptive services being specific activities that do not need a thread to preempt.

19. The system as recited in claim 15 wherein the itinerary running service comprises:

a. a preemption service enabling thread preemption;

b. standard preemptive services, the preemptive services being specific activities that can preempt the thread, the standard preemptive services using the preemption service for execution;

c. standard non-preemptive services, the non-preemptive services being specific activities that do not need a thread to preempt; and

d. an itinerary building service, the itinerary building service facilitating execution of itinerary specific program constructs.

20. An itinerary running service for executing an itinerarized thread in itinerary mode, the itinerarized thread comprising standard thread constructs and itineraries, the itineraries being lists of errands, the errands being small tasks needed to be performed during thread execution, the itinerary mode being a special thread execution scheme for executing the complete itinerary, the itinerary running service receiving the itinerary from an operating system scheduler, the itinerary running service comprising:

a. means for providing various preemptive and non-preemptive services to the itinerary, the preemptive services being specific activities that may preempt the thread to preempt, the non-preemptive services being specific activities that do not need the thread to preempt;

b. an itinerary building service providing pre-programmed standard errands and enabling specification of application specific errands, the standard errands being various preemptive and non-preemptive services provided by the system, the application specific errands being special program constructs built into the itinerary;

c. means for executing the errands in the itinerary sequentially; and

d. means for blocking the itinerary when one of the errands blocks.

21. The itinerary service as recited in claim 20 further comprising means for resuming execution of a previously blocked itinerary from the errand that blocked the itinerary, when the itinerary is scheduled back on the itinerary running service by the operating system scheduler.

22. A computer program product for executing a multithreaded application, the application having been programmed using itinerarized threads, the itinerarized threads comprising standard thread constructs as well as itineraries, the itineraries being lists of errands, the errands being functions performing specific tasks, the computer program product comprising:

a computer readable medium comprising:

a. program instruction means for compiling the application code;

b. program instruction means for scheduling the plurality of threads for execution; and

c. program instruction means for executing the plurality of threads in a manner that minimizes thread switching overheads and memory usage during the execution, the standard threads being executed using thread stacks associated with the threads as execution stacks, the itineraries being executed using kernel stack as their execution stack.

23. The computer program product as recited in claim 22 wherein the computer readable medium comprising program instruction means for executing the plurality of threads further comprises:

a. program instruction means for executing threads running standard thread constructs in normal mode, the normal mode of execution being thread execution in accordance with standard thread execution methodology;

b. program instruction means for preempting a thread running in normal mode in response to a request for running an itinerary on behalf of the thread, the preempted thread subsequently being run in itinerary mode, the itinerary mode being a special thread execution scheme for executing the complete itinerary; and

c. program instruction means for exiting the itinerary mode of execution when the complete itinerary corresponding to a thread has been executed, the thread being subsequently executed in normal mode.